

CSEN2061-DBMS-V Semester

Module 1 Notes

UNIT 1 Introduction to DBMS and Database Design

Introduction to DBMS: File system vs DBMS, advantages of DBMS, storage data, queries, DBMS structure

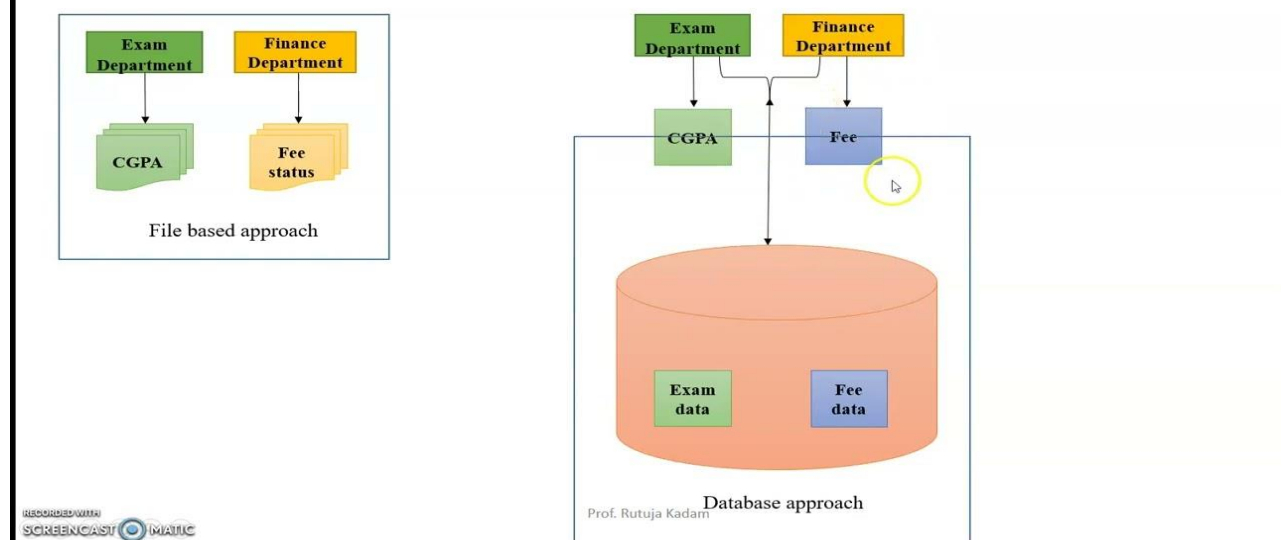
Types of Databases – Hierarchical, Network, Relational, Key-Value, Object Oriented, XML DB, Overview of File Structures in database

Data base Design: data models, the importance of data models.

E-R model: Entities, attributes and entity sets, relationship and relationship set, mapping cardinalities, keys, features of ER model, conceptual database design with ER model.

File system vs DBMS

□ File system and Database system approach



A file system is a **method for organizing and storing files on a storage device**, while a database is a **structured collection of logically related data, often managed by a database management system (DBMS), designed for efficient storage, retrieval, and management of large amounts of data and information of an application under consideration.**

Essentially, file systems provide aspects like storage and access to files, while databases offer more advanced features like data relationships, querying, and security.

Difference between file system and DBMS

File Management e.g. C++ or COBOL program	Database Management e.g. Oracle or Sybase
Small systems	Large systems
Relatively cheap	Relatively expensive
Few 'files'	Many 'files'
Files are files	Files are tables
Simple structure	Complex structure
Redundant data	Reduced redundancy
Chances of inconsistency	Consistent
Isolated data	Data can be shared
Little preliminary design	Vast preliminary design
Integrity left to application programmer	Rigorous inbuilt integrity checking
No security	Rigorous security
Simple, primitive backup/recovery	Complex & sophisticated backup/recovery
Often single user	Multiple users

Advantages and Disadvantages of DBMS

DBMS

Advantages

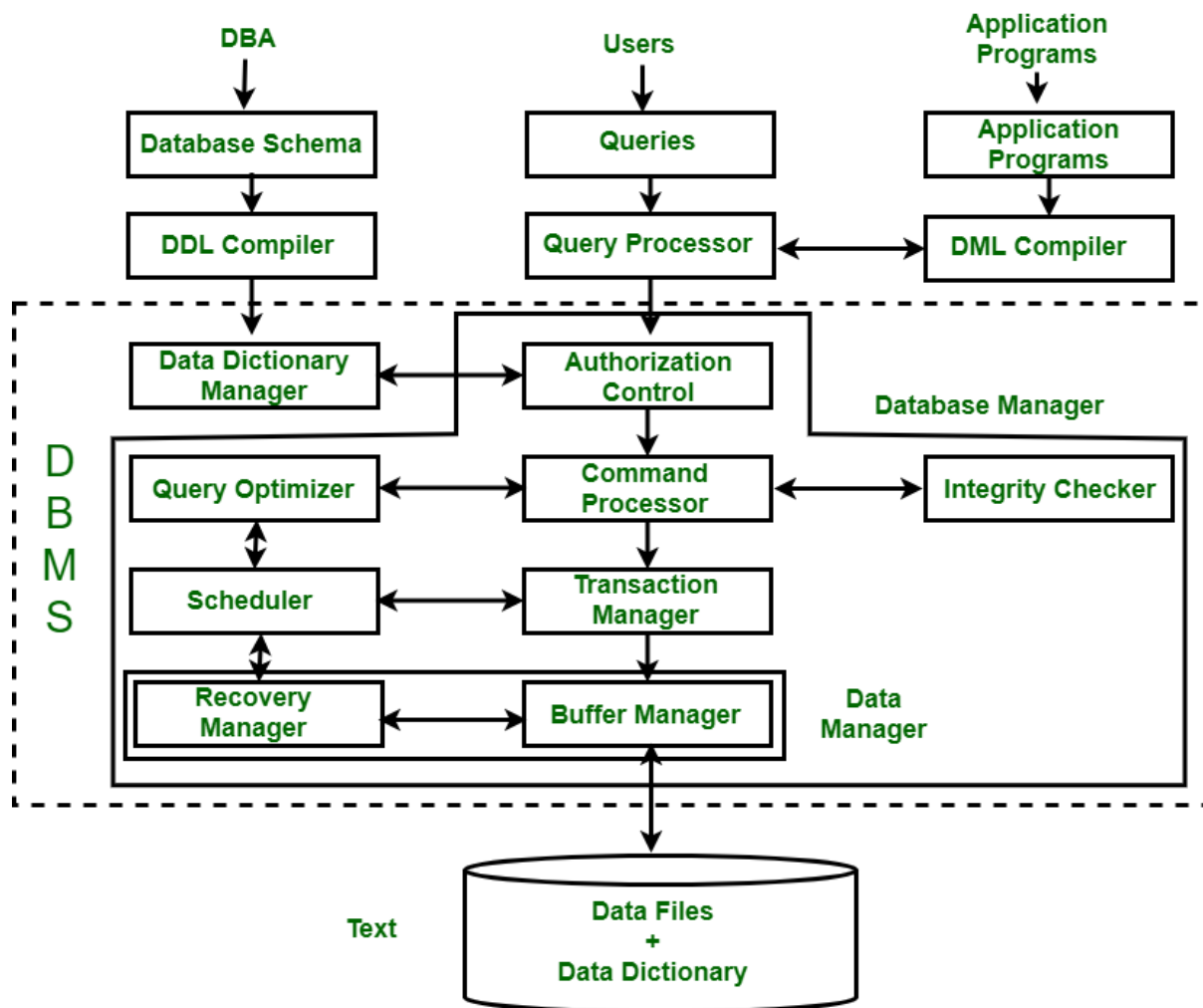
- Improved data security
- Easy data retrieval
- Minimum data inconsistency
- Improved decision making
- Better data sharing facility
- Improved data integration
- Good data back-up

Disadvantages

- Expensive
- Complex to operate
- Not beneficial for small firms
- Frequent upgrades



Structure of DBMS



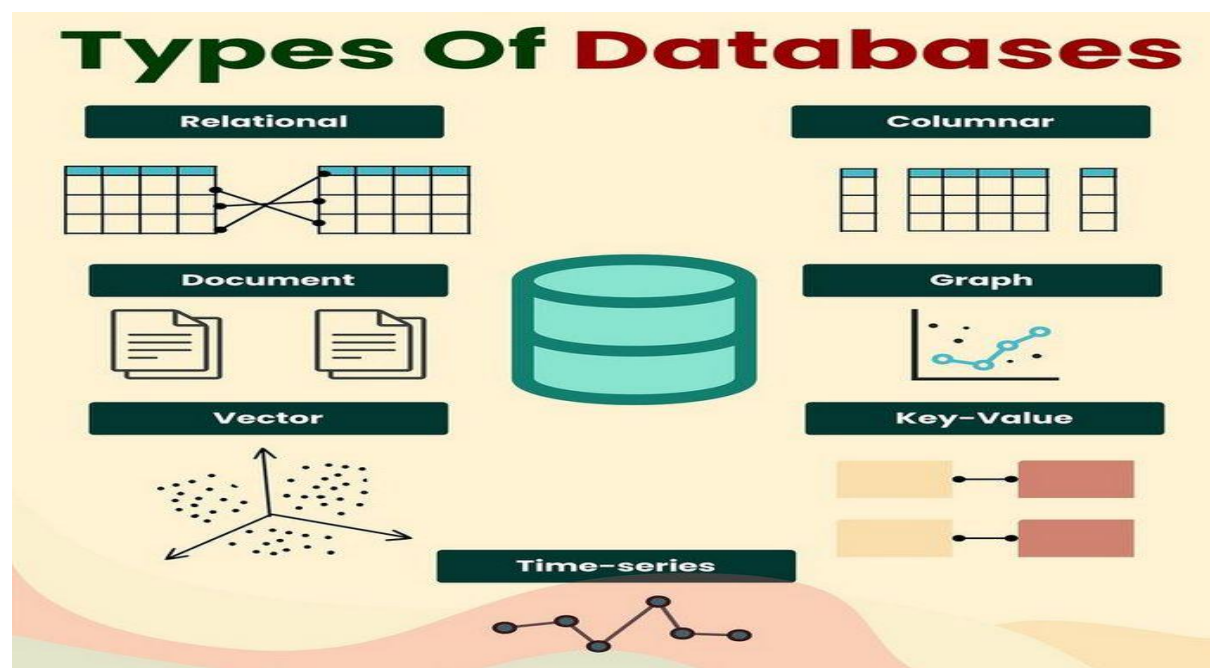
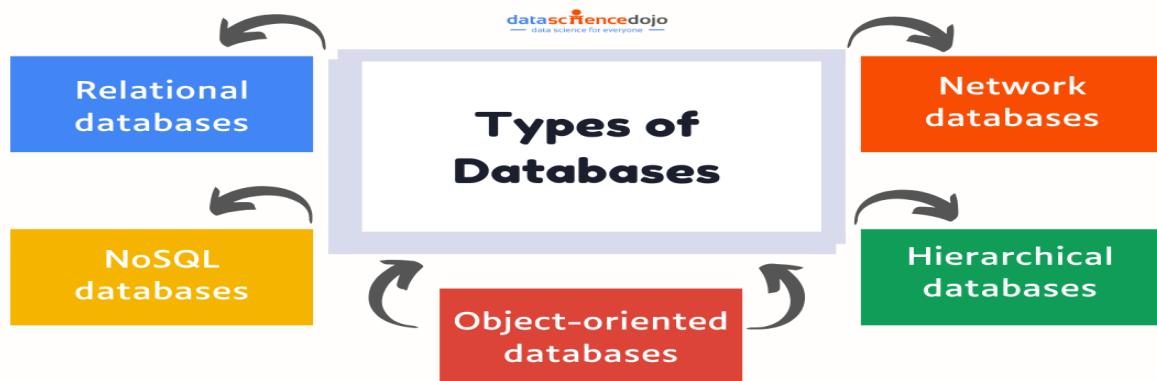
This figure represents the **functional architecture of a Database Management System (DBMS)**, showcasing how different components interact to manage data effectively and securely. The process starts with the **Database Administrator (DBA)** who defines the **Database Schema**, which is compiled by the **DDL (Data Definition Language) Compiler**. This information is stored and managed by the **Data Dictionary Manager**, which maintains metadata about the database. Users interact with the DBMS by submitting **queries**, which are handled by the **Query Processor**. Simultaneously, **Application Programs** access the database through the **DML (Data Manipulation Language) Compiler**.

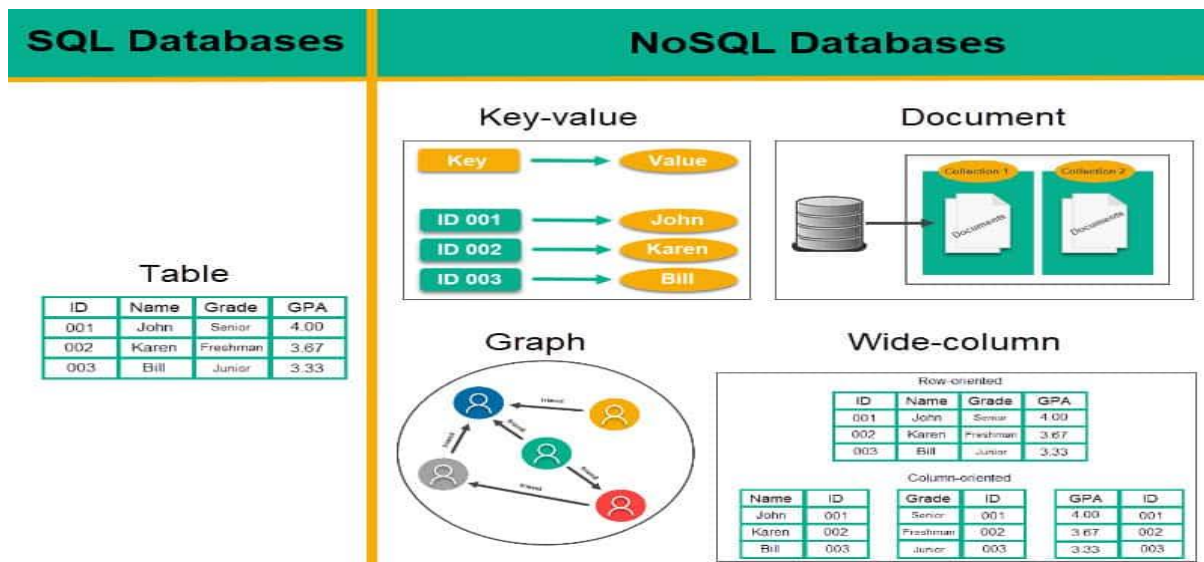
Security is ensured by the **Authorization Control** module, which verifies user permissions before passing commands to the **Command Processor**. The **Integrity Checker** ensures that all data modifications comply with integrity constraints. The **Query Optimizer** refines queries for efficient execution, while the **Scheduler** manages the order of query execution to avoid conflicts in multi-user environments. The **Transaction Manager** handles transaction execution, maintaining **ACID properties** (Atomicity, Consistency, Isolation, Durability) and coordinating with the **Buffer Manager**, which temporarily stores data in memory for faster access.

In case of failures, the **Recovery Manager** restores the system to a consistent state. The **Buffer Manager** and **Transaction Manager** work together to read from and write to the physical **Data Files and Data Dictionary**. The entire system is divided into three main subsystems: the **Query Processor**,

the **Database Manager**, and the **Storage/Data Manager**, all working cohesively to ensure data integrity, performance, and security in a DBMS environment.

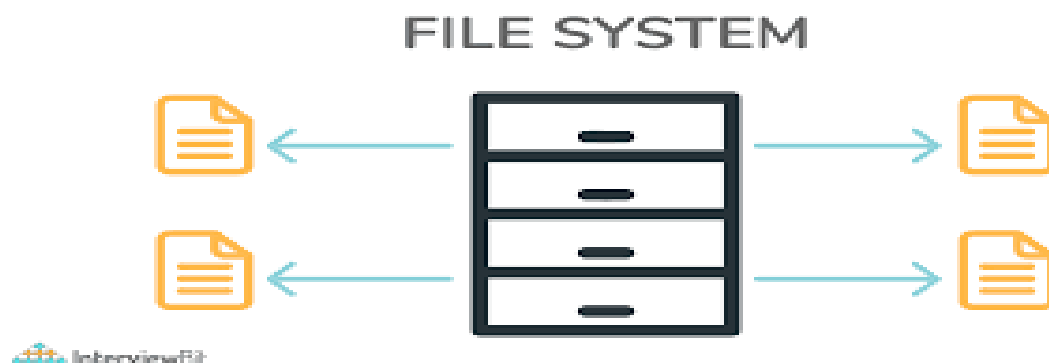
Types of Databases





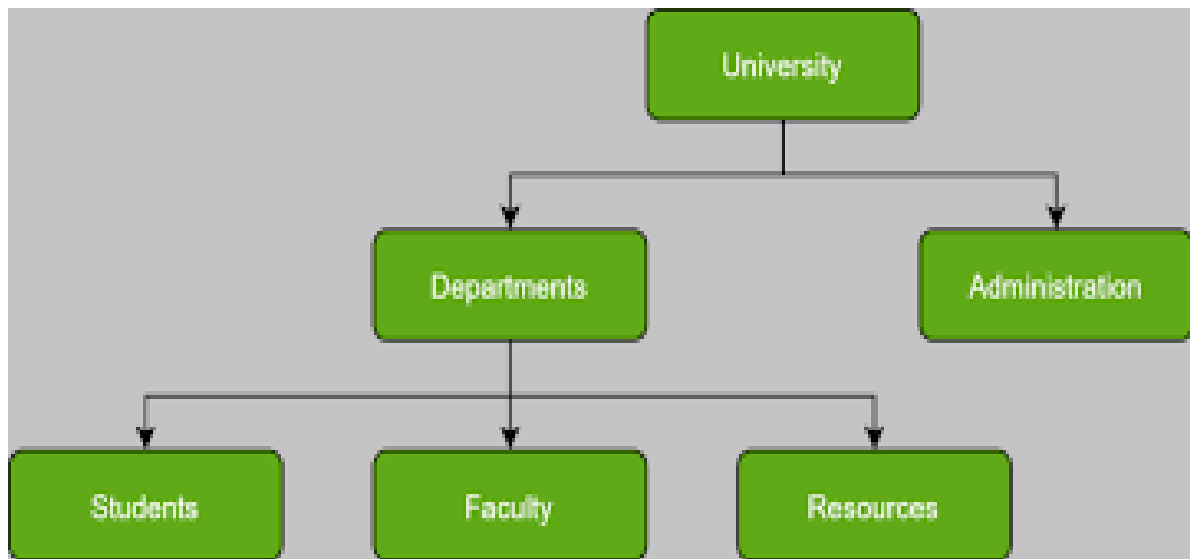
Different Types of Databases

Flat Files



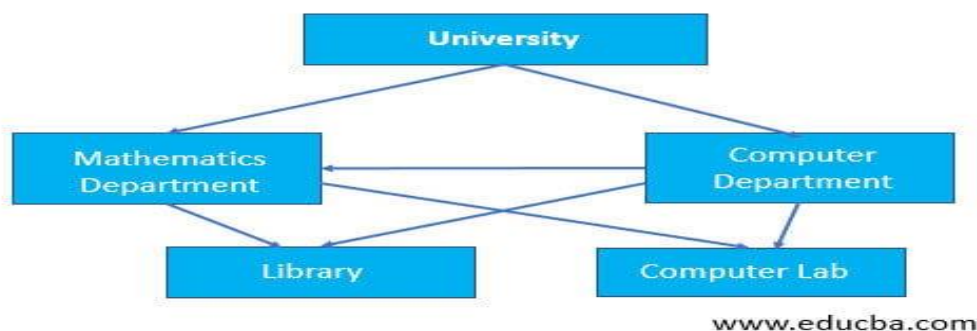
This figure illustrates the concept of a File System, where data is stored in individual, separate files managed directly by applications. The central cabinet represents the file storage, and the documents around it symbolize isolated files. Each file operates independently, leading to data isolation, meaning data in one file is not easily accessible or shareable with other files or applications. There is no sharing of data across files unless explicitly programmed, making it difficult to enforce consistency, prevent redundancy, and manage data efficiently. This lack of integration highlights one of the major limitations of file-based systems compared to modern DBMS.

Hierarchical Database



This figure illustrates a **hierarchical model** of a **University database system**, where data is organized in a **tree-like structure**. In this data gets organized in levels. At the top is the **University**, representing the root or main entity. It branches into two major sub-entities: **Departments** and **Administration**. The **Departments** entity further branches out to manage three specific components: **Students**, **Faculty**, and **Resources**. This indicates that student records, faculty information, and departmental resources are all organized under their respective departments. The hierarchy reflects a **parent-child relationship**, where each lower-level entity depends on and is accessed through its parent. This structure shows **data dependency** and **rigid links**, which are typical of hierarchical databases but can lead to **limited flexibility** when changes are required.

Network Database



This figure illustrates a **network data model** of a **University system**, where entities like the **Mathematics Department** and **Computer Department** are connected to the **University** and also share links to common resources such as the **Library** and **Computer Lab**. The model uses **multiple links** to represent **many-to-many relationships**, enabling flexible access and efficient data sharing among departments and resources. However, a **major problem** with this model is its **structural complexity**—as the number of entities and links increases, the database becomes harder to design, update, and maintain. Additionally, **navigating through multiple linked paths** requires complex programming and increases the risk of data inconsistency if not properly managed.

Relational Database

stuId	lastName	firstName	major	credits
S1001	Smith	Tom	History	90
S1002	Chin	Ann	Math	36
S1005	Lee	Perry	History	3
S1010	Burns	Edward	Art	63
S1013	McCarthy	Owen	Math	0
S1015	Jones	Mary	Math	42
S1020	Rivera	Jane	CSC	15

Figure 1.1A The Student Table

classNumber	facId	schedule	room
ART103A	F101	MWF9	H221
CSC201A	F105	TuThF10	M110
CSC203A	F105	MThF12	M110
HST205A	F115	MWF11	H221
MTH101B	F110	MTuTh9	H225
MTH103C	F110	MWF11	H225

Figure 1.1C The Class Table

facId	name	department	rank
F101	Adams	Art	Professor
F105	Tanaka	CSC	Instructor
F110	Byrne	Math	Assistant
F115	Smith	History	Associate
F221	Smith	CSC	Professor

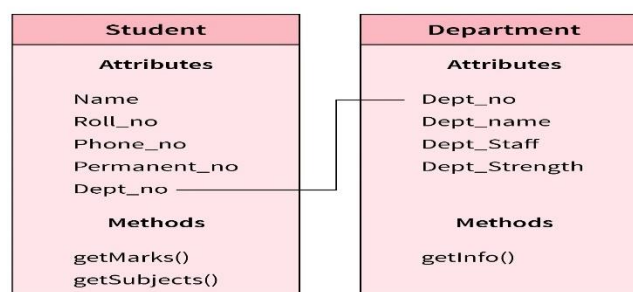
Figure 1.1B The Faculty Table

stuId	classNumber	grade
S1001	ART103A	A
S1001	HST205A	C
S1002	ART103A	D
S1002	CSC201A	F
S1002	MTH103C	B
S1010	ART103A	
S1010	MTH103C	
S1020	CSC201A	B
S1020	MTH101B	A

Figure 1.1D The Enroll Table

This figure illustrates a **relational database** for a university system, where data is organized into multiple interrelated tables: **Student**, **Faculty**, **Class**, and **Enroll**. Each table represents a specific entity and contains unique identifiers (like `stuId`, `facId`, and `classNumber`) used as **primary keys**. These keys are referenced as **foreign keys** in other tables to establish relationships—for example, `stuId` in the **Enroll** table links to the **Student** table, and `facId` in the **Class** table links to the **Faculty** table. This structure allows for **efficient data retrieval**, **data integrity**, and **minimal redundancy** by enabling complex queries across the connected tables, which is a key advantage of the relational database model.

Object-oriented database




SCALER
Topics

This figure represents an **Object-Oriented Database (OODB)** model, where data is stored as **objects**, similar to object-oriented programming concepts. The model shows two classes: **Student** and **Department**, each containing **attributes** (data fields) and **methods** (functions/operations). For example, the **Student** object has attributes like Name, Roll_no, and Dept_no, and methods like getMarks() and getSubjects(). The **Department** object includes attributes such as Dept_no, Dept_name, and a method getInfo(). The **link between Dept_no in Student and Department** shows a relationship between the objects. This approach allows for **encapsulation, inheritance, and reusability**, making it well-suited for applications with complex data types such as CAD, multimedia, and real-time systems.

Key-Value database

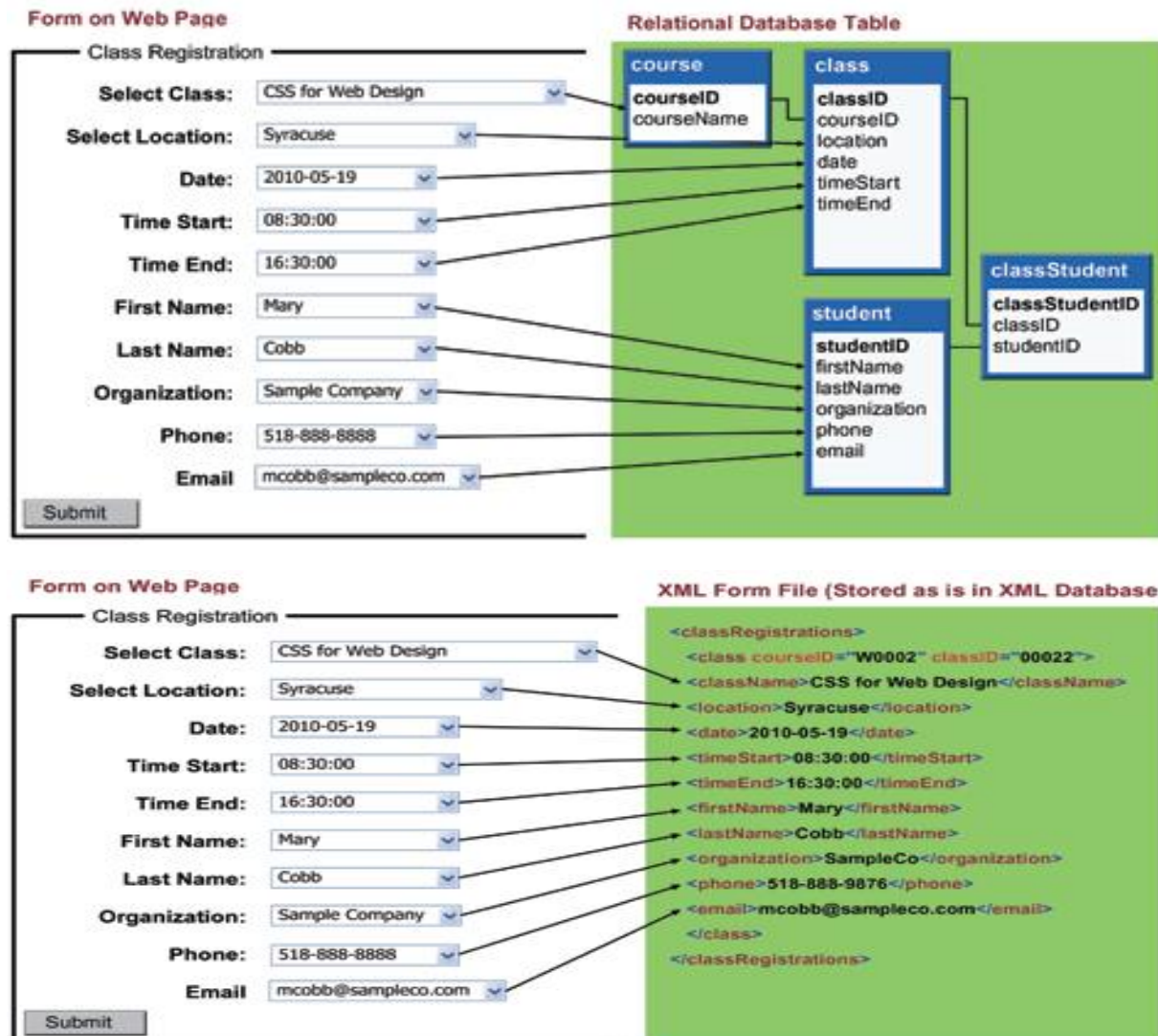
Roll. No.	First Name of Student	Last Name of Student	Course code
0111	Adam	Gilchrist	A100
0222	Adam	Peter	B50
0333	John	Gilchrist	C80



The diagram shows a red rectangular box highlighting the 'First Name of Student' and 'Last Name of Student' columns of the table. Two arrows originate from the bottom of this box: one points to the text 'Non-key Attribute' and the other points to the 'Last Name of Student' column header.

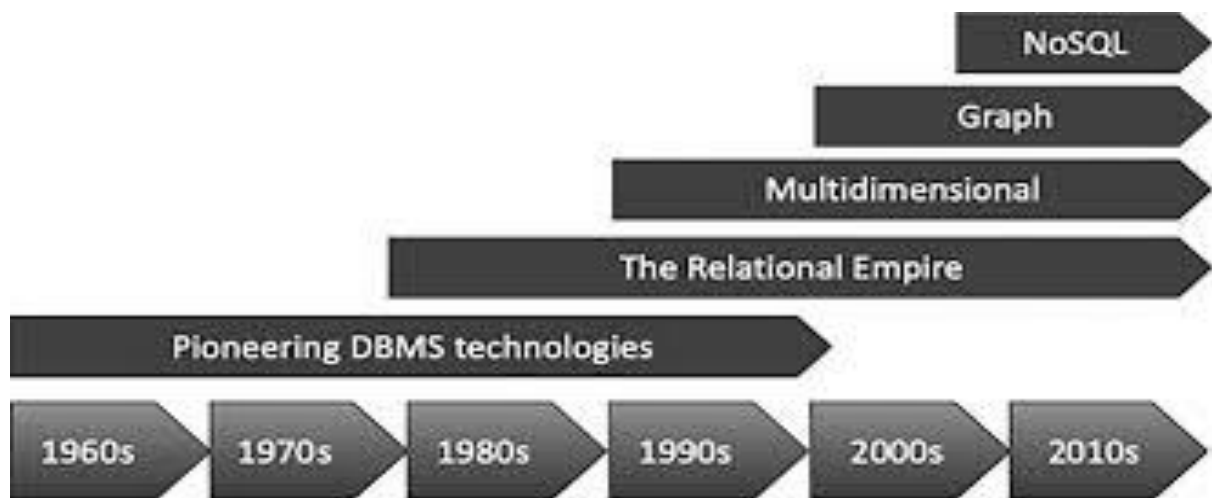
This figure illustrates the concept of a **Key-Value Database**, where each record consists of a **unique key** and its associated **value(s)**. In this example, the **Roll No.** acts as the **key**, uniquely identifying each student record. The associated values include non-key attributes like **First Name**, **Last Name**, and **Course Code**. A key-value database stores data as a collection of key-value pairs, making it highly efficient for **quick lookups, insertions, and updates** using the key. This model is widely used in applications requiring **fast access to data**, such as caching, session storage, and real-time recommendations. However, it lacks structure for complex relationships and is not ideal for relational queries.

XML database



This figure demonstrates how data entered through a **web form** (such as class registration) is stored in a **Relational Database**. However, in the context of an **XML Database**, this same data would be stored and represented in **XML format**, where each data entry is structured using **tags** to define elements and hierarchy. Unlike relational databases that use tables with fixed schemas, XML databases store data in a **tree-like, hierarchical structure**, making it flexible and suitable for semi-structured data like web forms, documents, and data exchange. For example, the course, class, and student data would be wrapped in XML tags like `<courseName>`, `<firstName>`, or `<email>`. A key benefit of XML databases is that they are **platform-independent**, easy to transport across systems (especially in web services), and support **nesting and flexible schemas**, but they can be **less efficient** for complex queries compared to relational databases.

Evolution of Database Systems

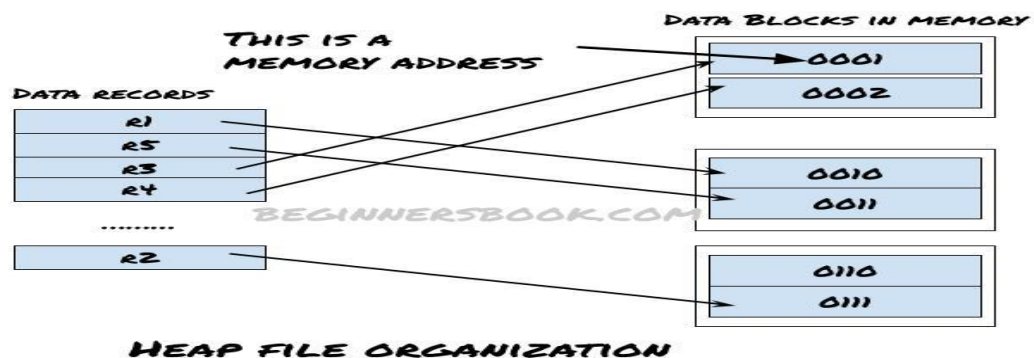


Overview of File Structures in Database



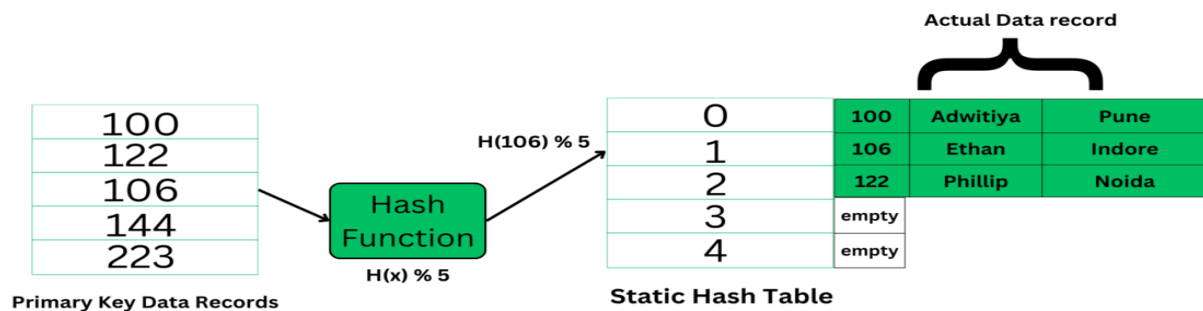
This figure represents different **file organization structures** in a **DBMS (Database Management System)**, which define how data is stored, accessed, and managed in secondary storage.

Heap File Structure



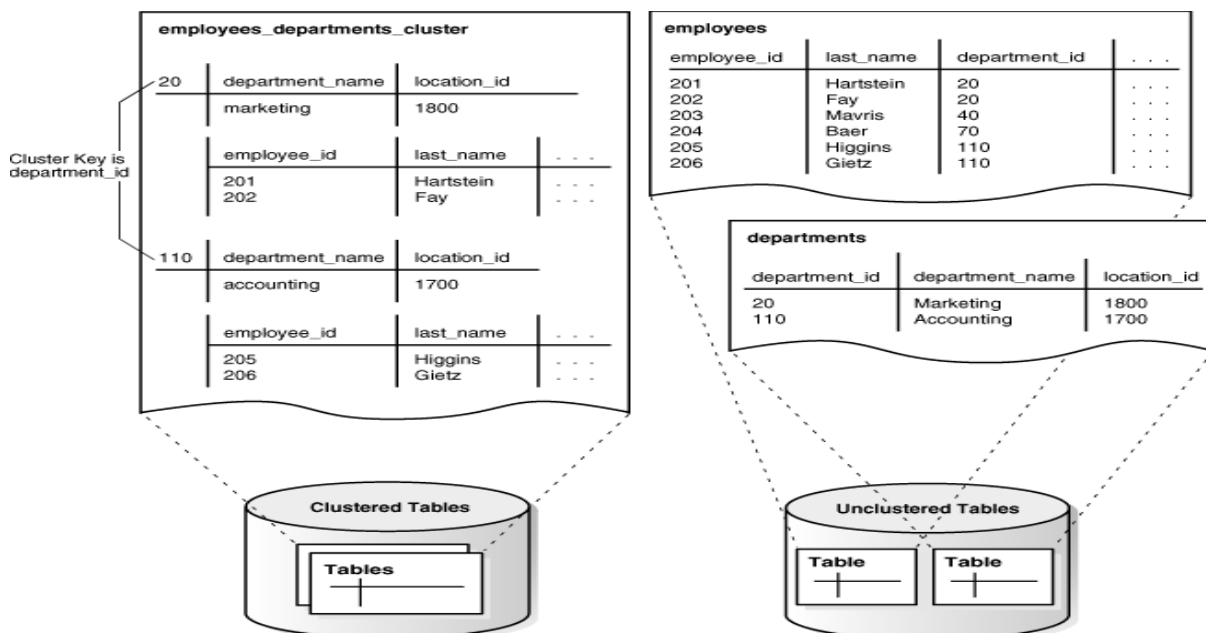
Heap File System: Records are stored in no particular order. It is simple and efficient for bulk insertions but inefficient for searches.

Hash File Structure



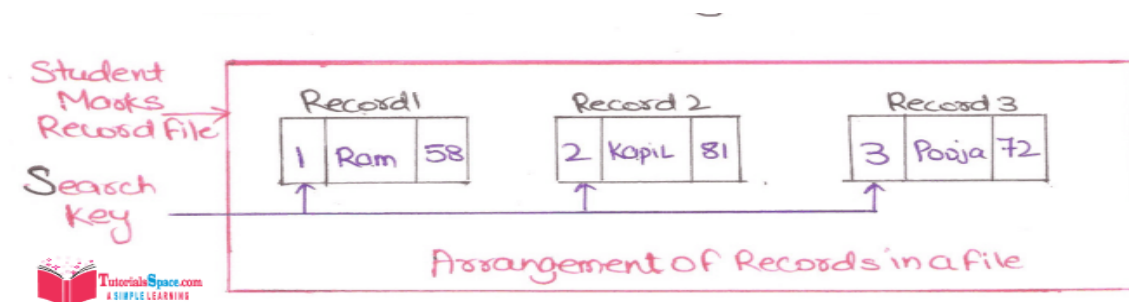
Hash File System: Uses a hash function to determine the location of records. It is ideal for equality searches but not suitable for range queries.

Clustered File Structure

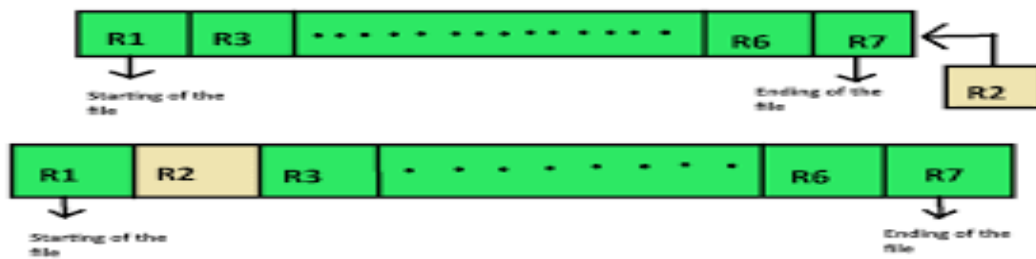


Clustered File System: Stores related records from different tables together for faster join operations and improved I/O efficiency

Sequential File Structure



Records Arrangement In A File



Sequential File System: Records are stored in a sorted order based on a key field. It offers fast reading and searching for sorted data, but makes insertion and deletion slower.

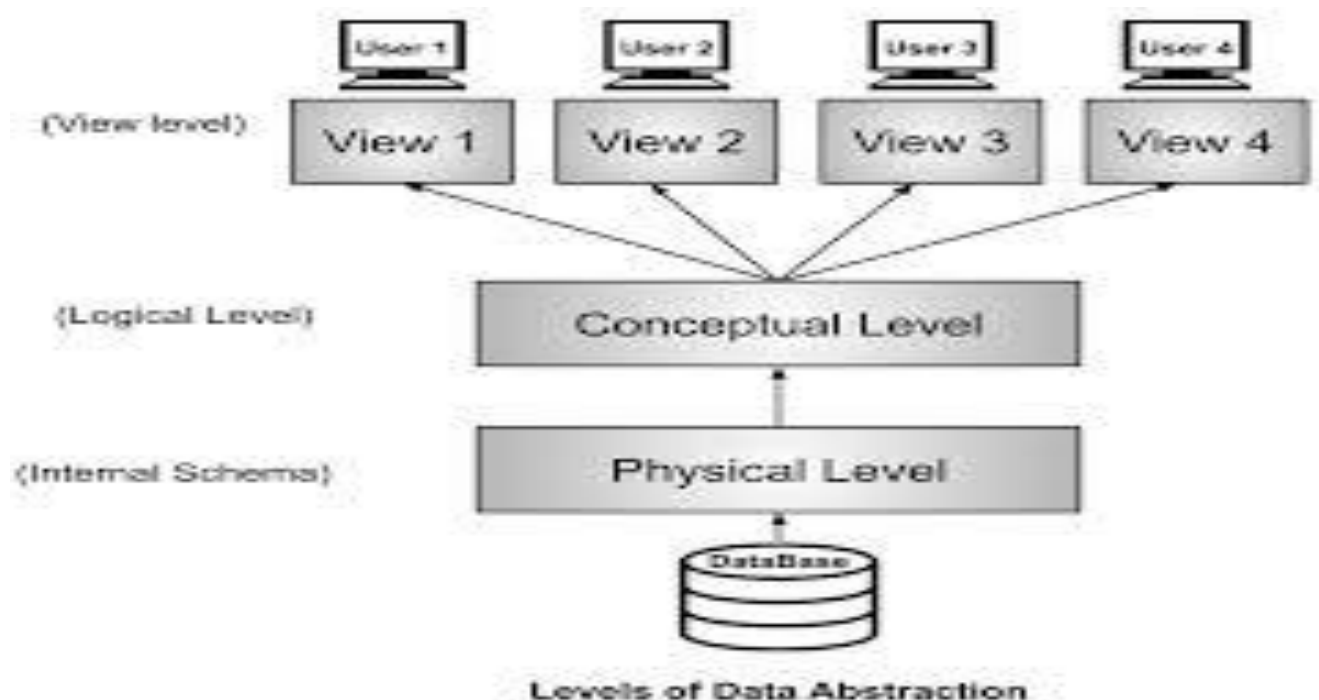
Database Design

Database design is the process of structuring and organizing data logically and efficiently to support storage, retrieval, and manipulation within a database system. It involves defining the **schema**, which includes tables, fields (attributes), data types, and the relationships between tables. The goal of database design is to create a system that is **efficient**, **redundant-free**, **scalable**, and maintains **data integrity**(correctness).

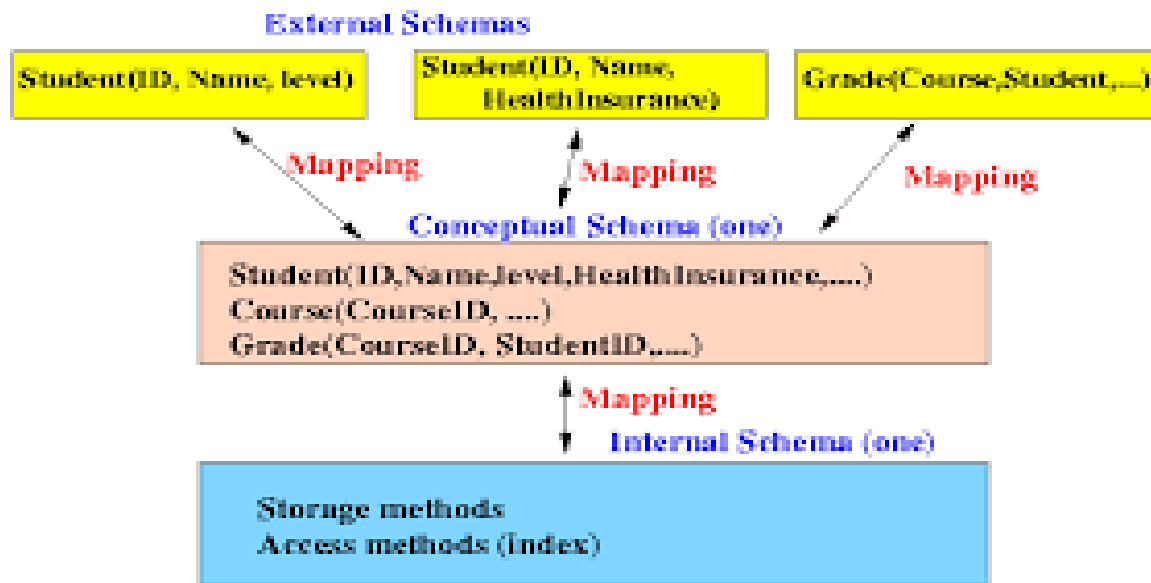
Key steps include

- **requirements analysis**,
- **conceptual design** (using ER models),
- **logical design** (defining relational schema), and
- **physical design** (optimizing storage and indexing). A well-designed database ensures accurate data management, easy updates, and smooth interaction with applications.

Three-Schema Architecture of DBMS

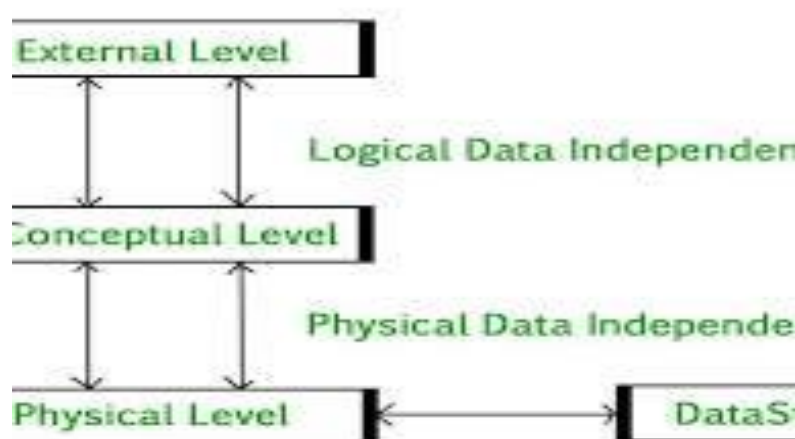


The figure illustrates the Three-Schema Architecture of a database system, which **provides data abstraction (hiding unnecessary details) at three levels**: View Level, Conceptual Level, and Physical Level. At the View Level, different users (User 1 to User 4) interact with personalized views of the database, hiding complexities and showing only relevant data. The Conceptual Level (Logical Schema) defines the overall database structure and the relationships between data without focusing on how it is stored. The Physical Level (Internal Schema) deals with the actual storage of data on hardware, specifying how data is stored, indexed, and accessed efficiently.



This layered architecture allows for **data independence**, meaning changes at the lower level do not affect higher levels, as these changes are automatically adopted by higher levels through mapping. This is improving the flexibility, security, and maintainability of the database system.

Data Independence



Data independence is the ability to modify the database schema at one level without affecting the schema at the next higher level, ensuring flexibility and minimal disruption to applications. It is of two types:

- **Physical Data Independence**, which allows changes in the internal storage or accessing methods without impacting the higher schemas, and
- **Logical Data Independence**, which enables changes in the logical structure (like adding or modifying tables and attributes) without altering users' views or application programs.

Conceptual Design (ER Modelling)

An Entity-Relationship (ER) model is a **high-level** data model that represents the logical structure of a database, depicting entities, their attributes, and the relationships between them. It is a pictorial representation of the design of the database to help clients understand the database design clearly.

There are three components of the ER Model. They are,

- **Entities:**

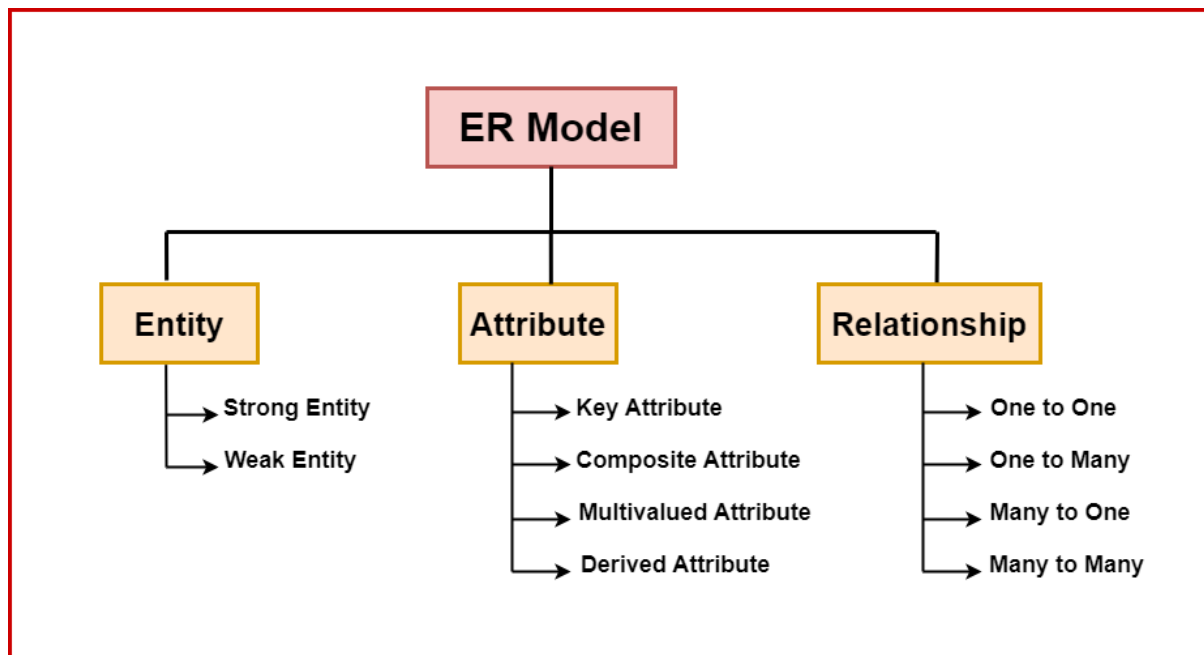
These represent **real-world objects** or concepts about which data is stored (e.g., a student, a course, a product).

- **Attributes:**

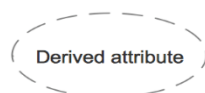
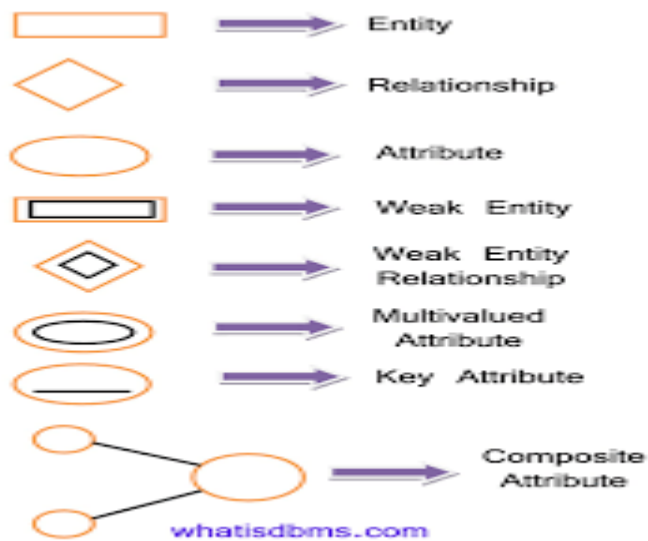
These are the **properties or characteristics of an entity** (e.g., a student's name, ID, or address).

- **Relationships:**

These define how entities are associated with each other (e.g., a student enrolls in a course)



The following is the list of symbols required to draw an ER Diagram to show the design of the database of the application under consideration.

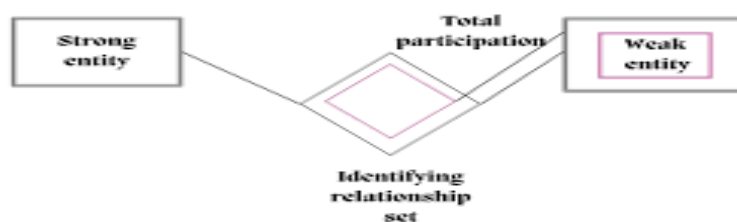


ENTITIES

1. Strong Entity Example



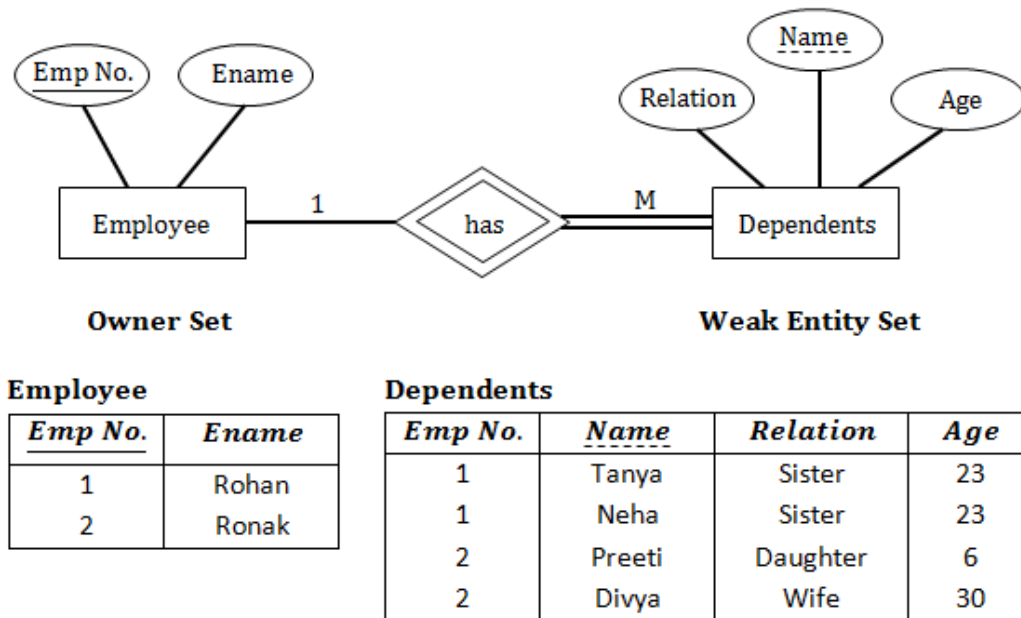
2. Weak Entity and Identifying Relationship



Always, a weak entity should be associated with its strong entity **through identifying a relationship**. The identifying relationship is shown by a **diamond-within-diamond symbol**. Further, a weak entity set **participates totally** in the relationship.

Examples for Weak Entity and Identifying (Weak) Relationship

Example 1



In the above example, dependents are to be grouped based on the employee. So, Dependents is a weak entity, and Employee is the strong entity

Note: Always, a weak entity(Dependent) should be associated with its strong entity (Employee) through identifying a relationship. The identifying relationship is shown by a diamond-within-diamond symbol. Further, a weak entity set participates totally in the relationship.

Example 2

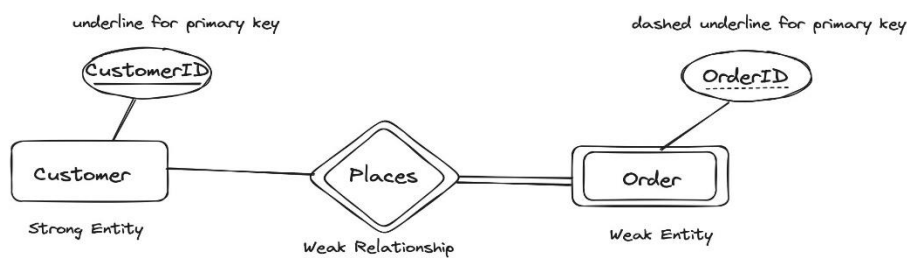


In the above example, courses are to be grouped based on the department. So, Courses is a weak entity, and Department is the strong entity

Note: Always, a weak entity(course) should be associated with its strong entity (Department) through identifying a relationship. The identifying relationship is shown by a diamond-within-diamond symbol. Further, a weak entity set participates totally in the relationship.

A few more examples of Weak Entity Set

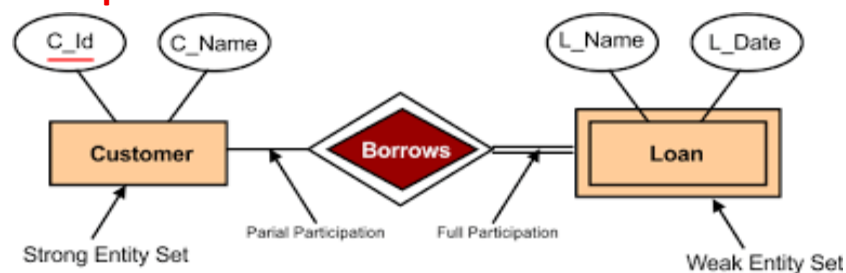
Example 3



Example 4

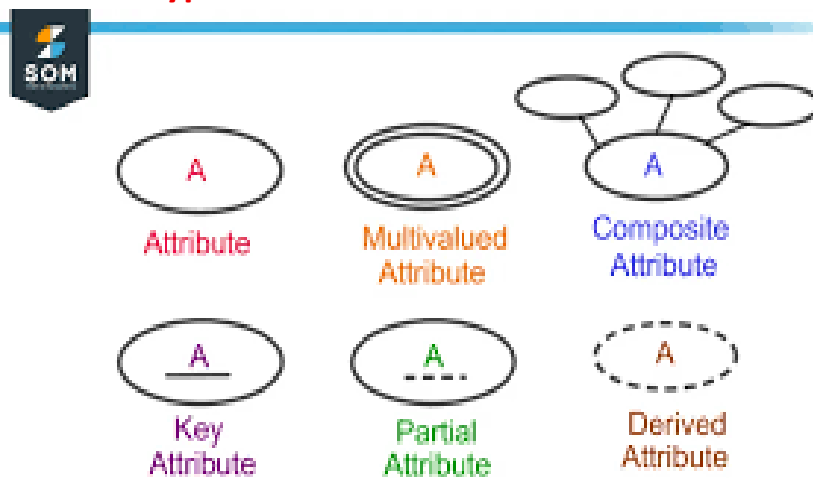


Example 5

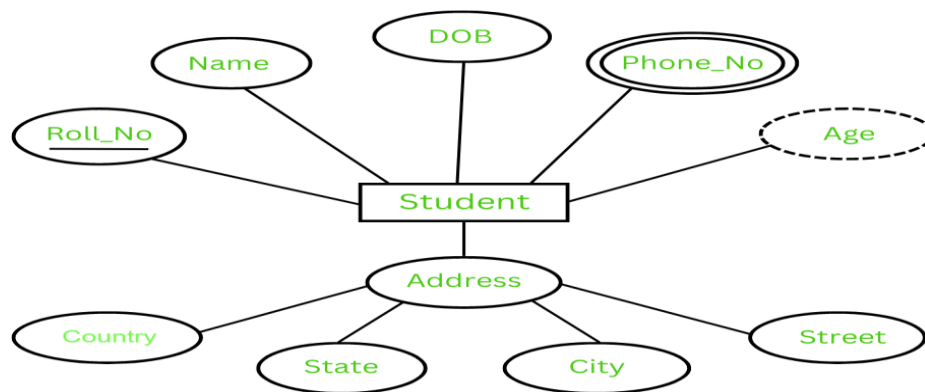


ATTRIBUTES

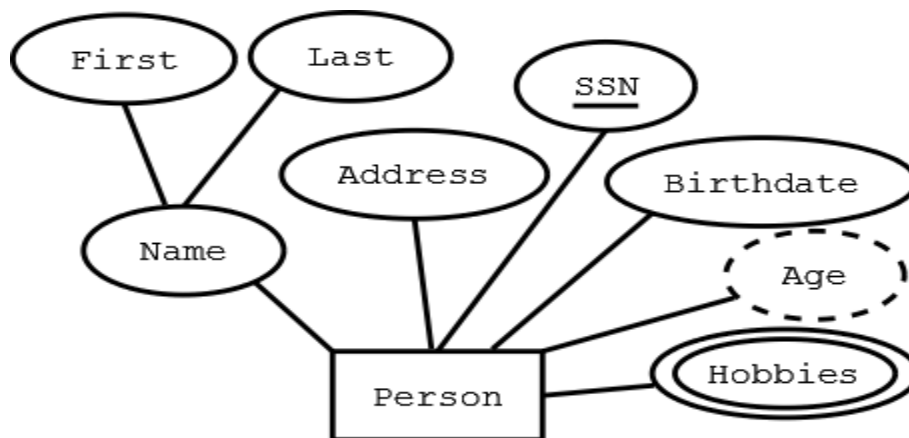
Different types of attributes



Example 1



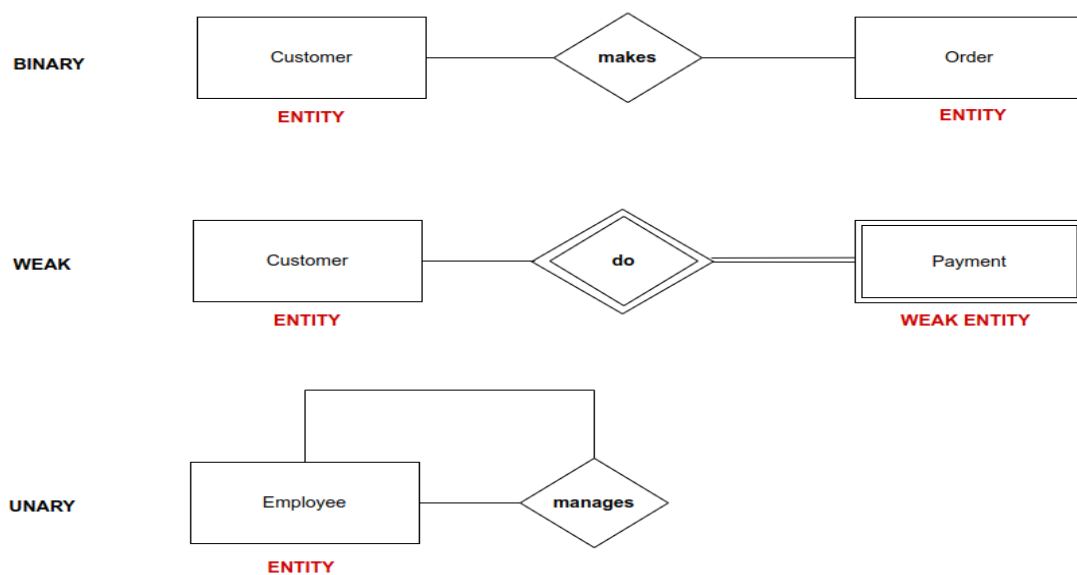
Example 2



RELATIONSHIPS

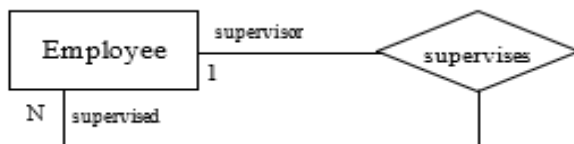
- Types
- Cardinality
- Participation

Types of relationships

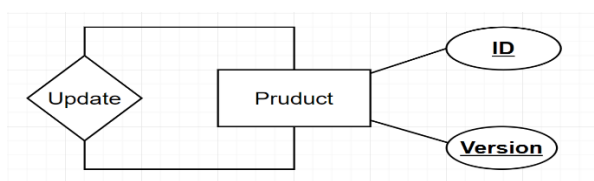


Note: **Unary relationship** is also known as a **Recursive Relationship**. In this relationship single entity set participates in the relationship more than once in different roles. Examples of a recursive relationship

Example 1



Example 2

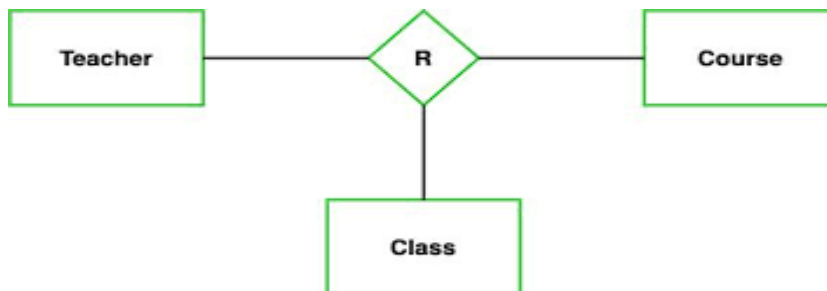


Example 3

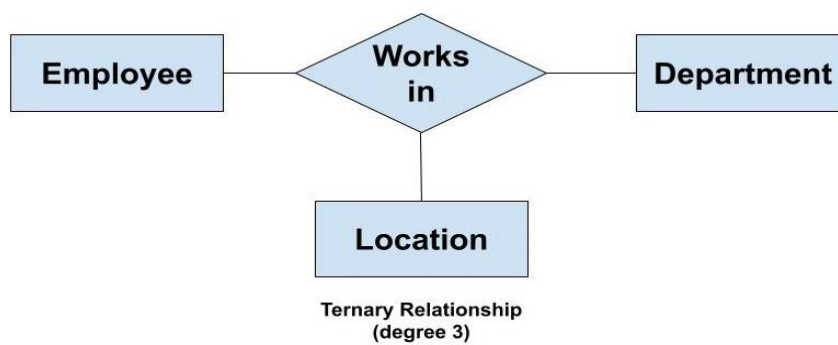
The teacher acts as a Mentor and Tutor
Draw it as a unary or recursive relationship.

Ternary

Example1



Example2

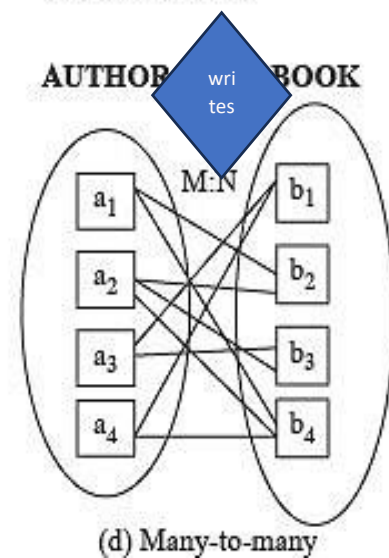
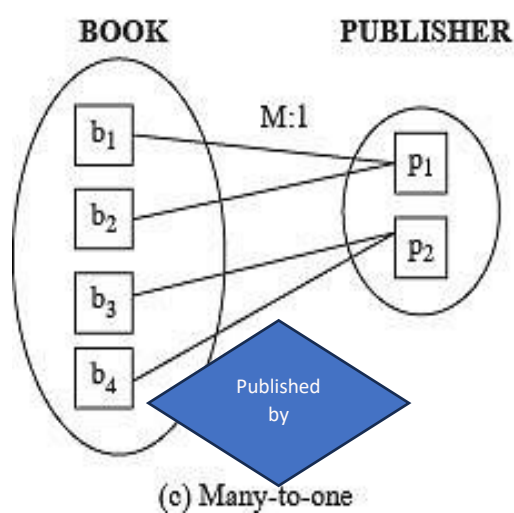
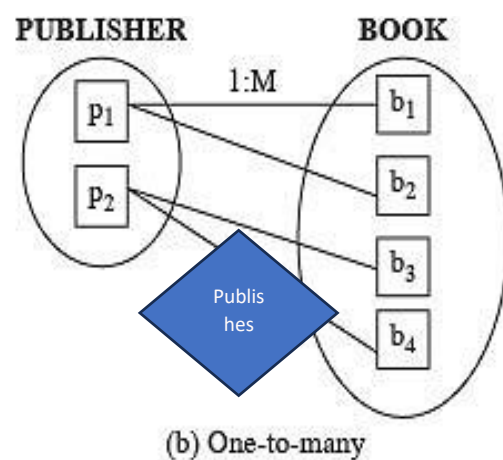
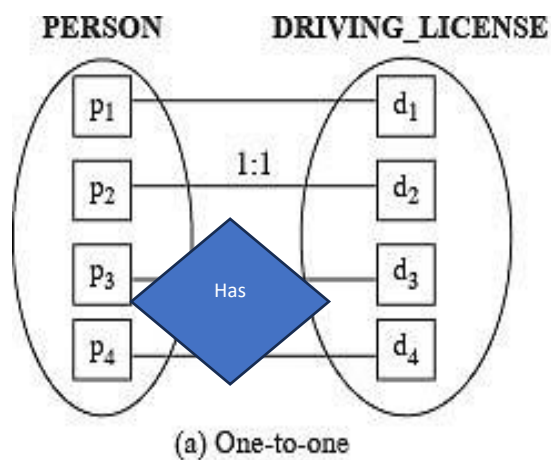


Cardinality of relationships

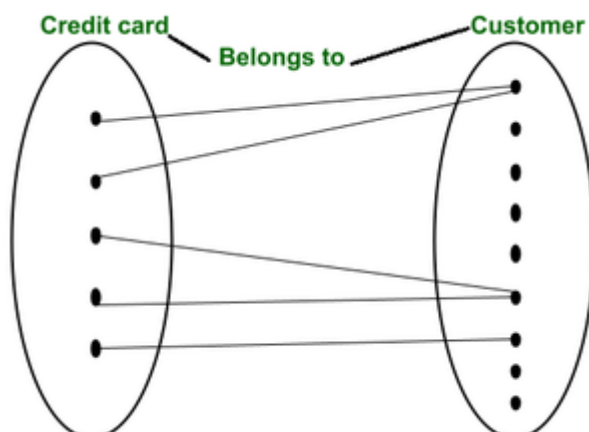
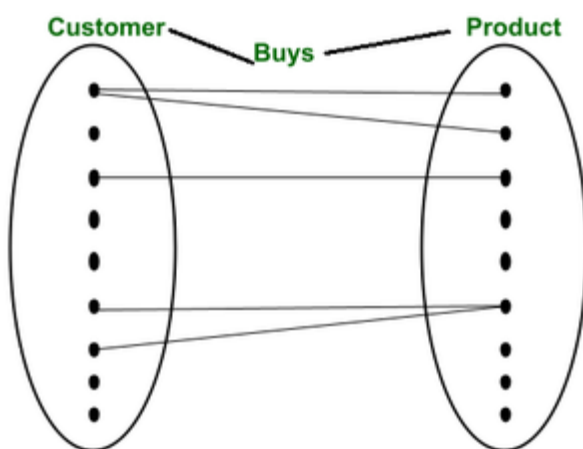
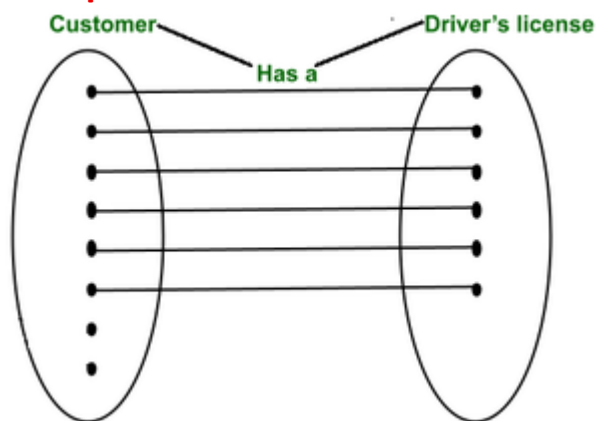
It refers to the number of instances of one entity that can be associated with instances of another entity. Four types of cardinality relationships are possible, as shown below.

- 1-to-1
- 1-to-M
- M-TO-1
- M-TO-N

Example 1

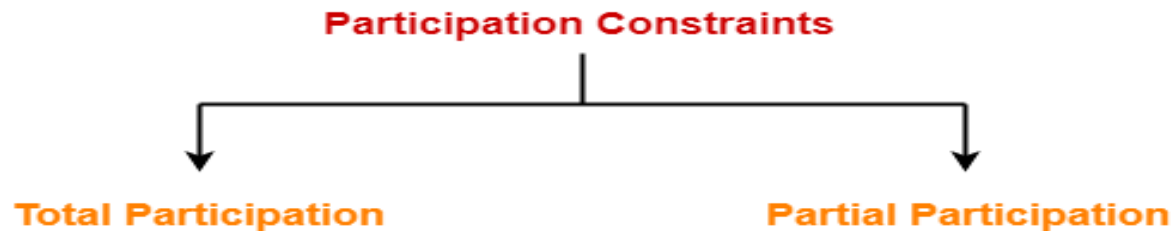


Example 2

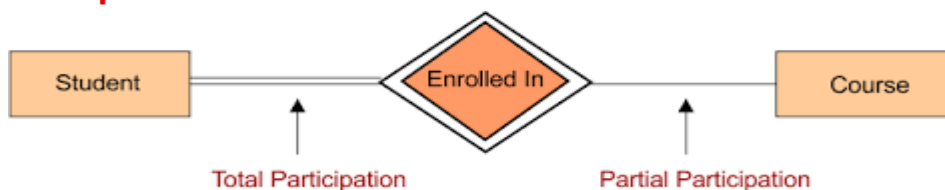


An entity participation in a relationship

It specifies whether all entities or only some entities of an entity set participate in a given relationship. The earlier one is called **TOTAL Participation**, and the next one is called **PARTIAL Participation**.



Example1



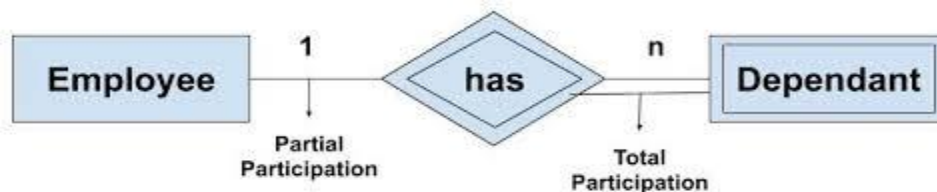
Example2



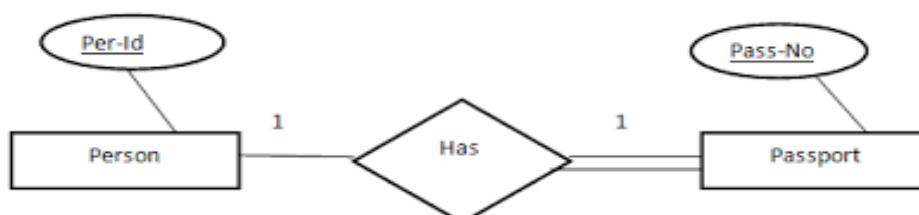
Example3



Example 4



Example 5



ER Modelling (Conceptual Design) Scenarios

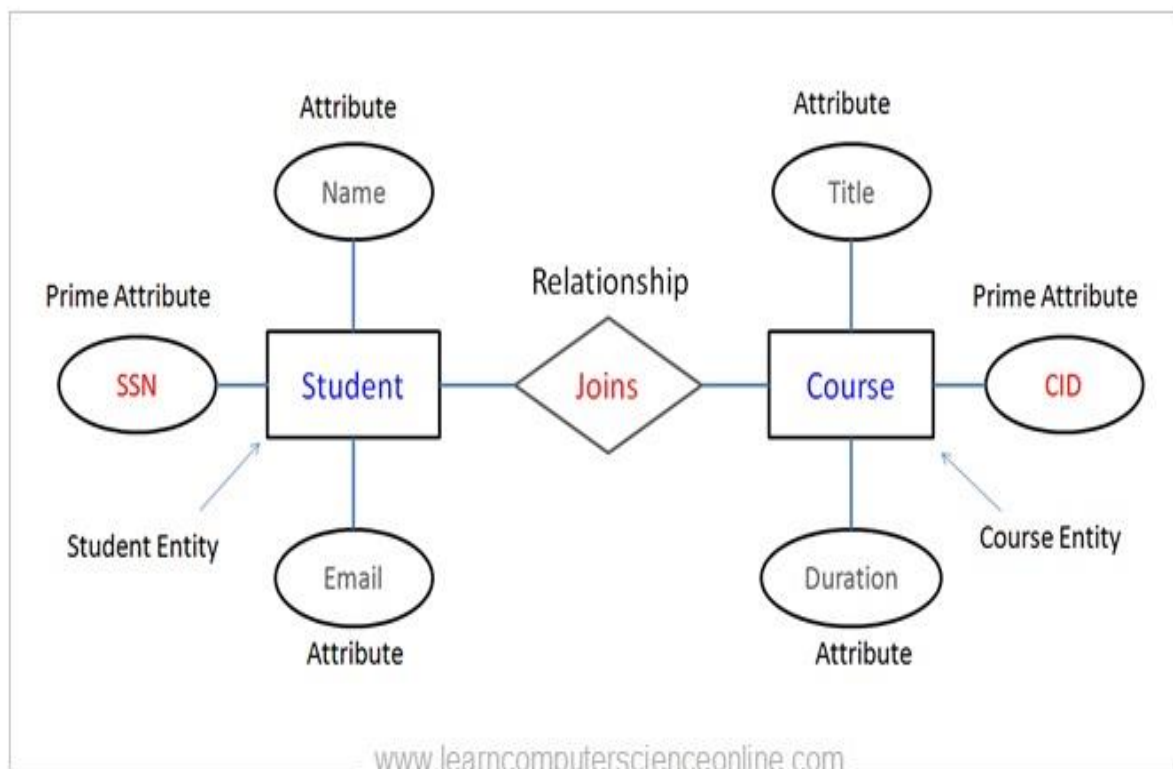
Examples of ER Modelling

Example 1 (Scenario) (Student joins the Course)

Student joins/registers Course. Student is described by ssn/regno, name, and email. The course is described by CID, title, and duration. Draw the ER diagram for this scenario.

Example 1 ER Diagram

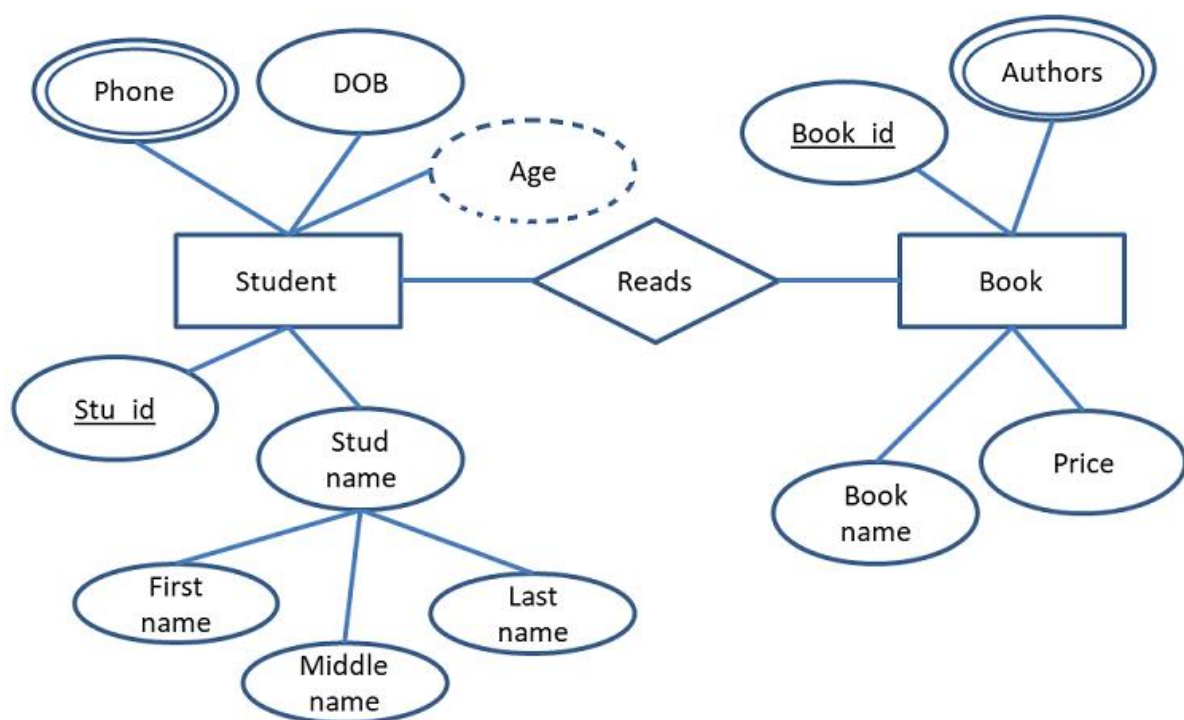
Entity Relationship Diagram (ERD)



Example 2 (Scenario) (Student reads the Book)

Student reads Book. Student is described by stu_id, stu_name, phone, dob. Further stu_name has three parts, namely, first name, middle name and last name. Student has got more than one phone number. Age is derived from dob. Book is described by book_id, book_name, authors, and price. Book can have more than one author. Draw the ER diagram for this scenario.

Example 2 ER Diagram



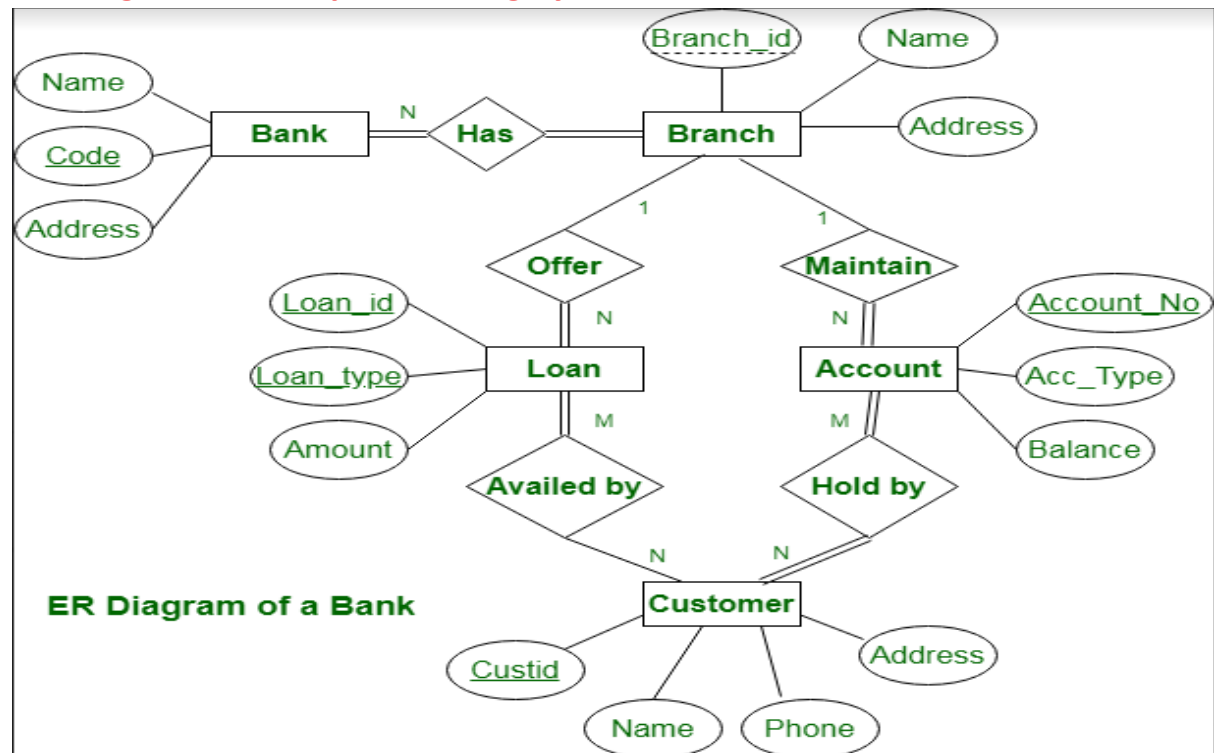
Exercises on ER Modelling

Exercise 1 (Scenario) (Simple Banking System)

A particular branch of a bank is offering a bank account and loan option to customers as per the bank's rules. The customer holds a bank account and avails a loan from the bank. A bank is described by bankcode, name, and address. A bank branch is specified by branch_id, name, and address. Account is specified by account_no, account_type and balance. Loan is specified by loan_id, loan_type, and amount of loan. Customer is specified

by customerid, customername, phone and address. Draw the ER diagram for this scenario, along with cardinality and participation aspects.

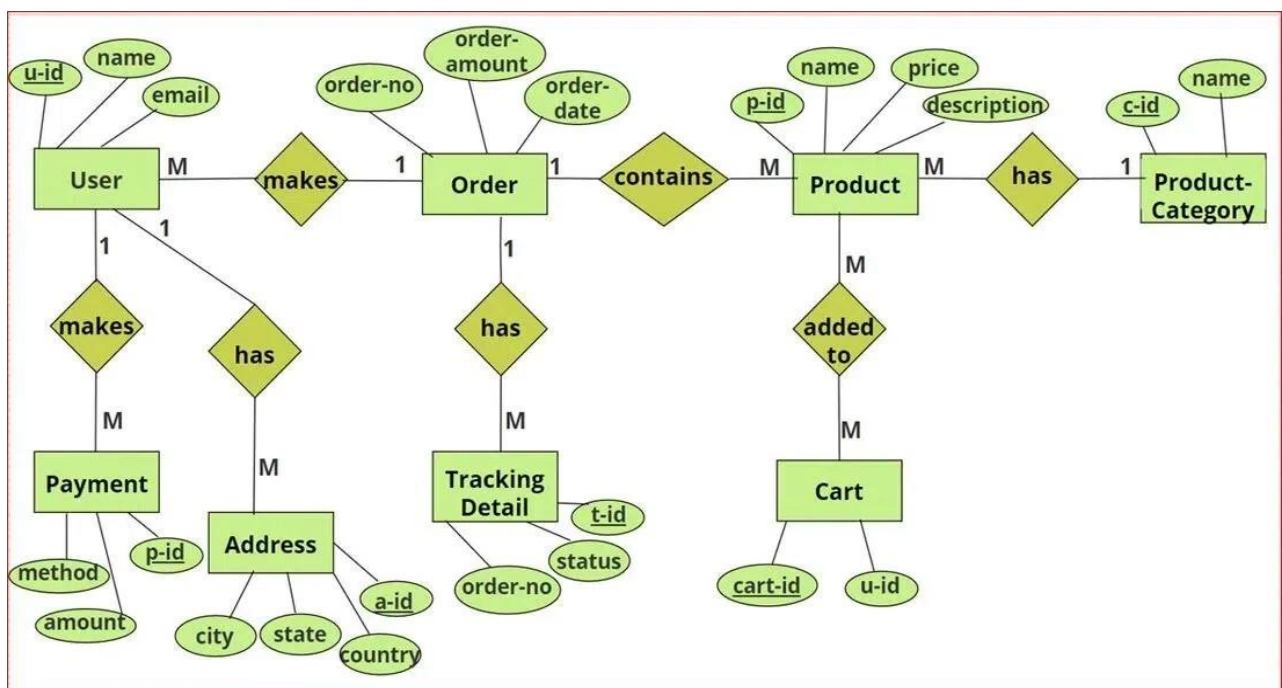
ER Diagram for Simple Banking System



Exercise 2 (Scenario)(Simple Online Shopping System)

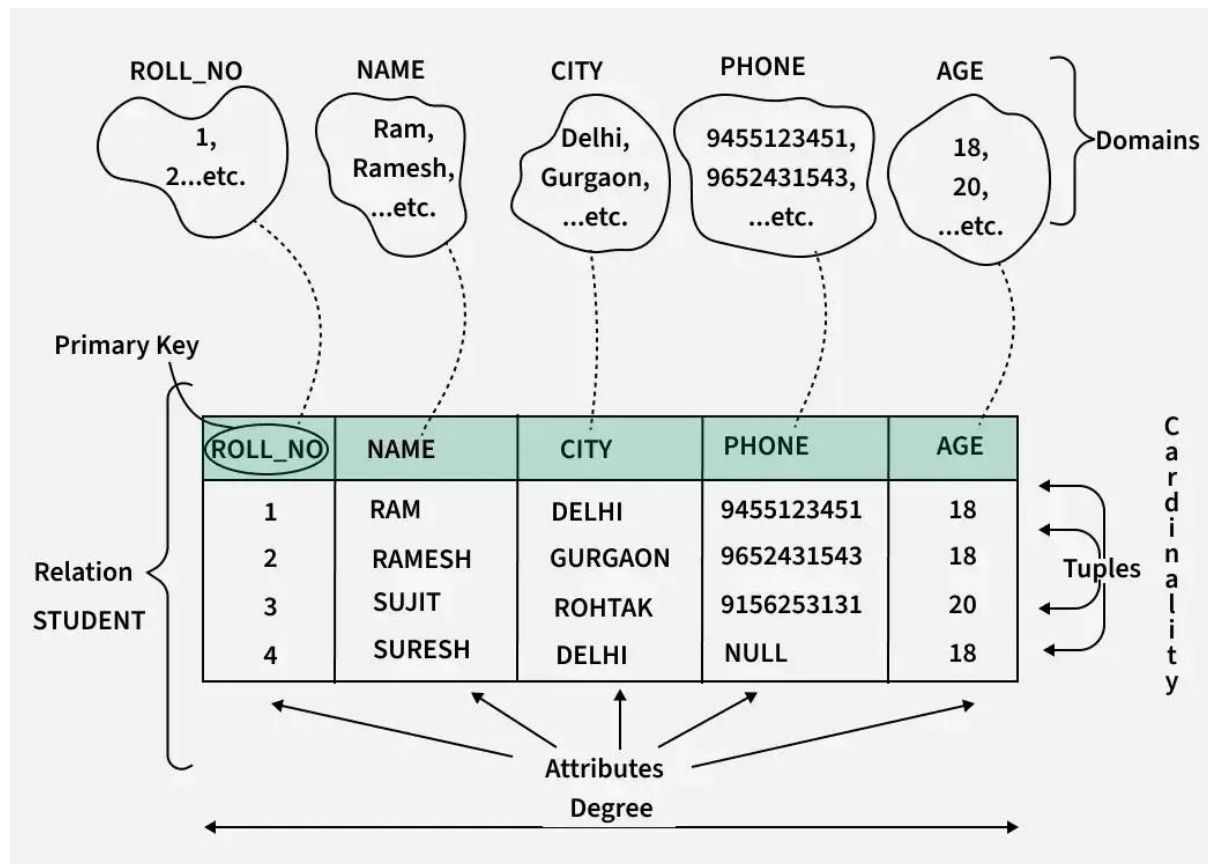
The user makes an order containing products of different categories online. The user gives the address for delivery and makes the payment online. User (u-id, name, email), Order (order-no, order-amount, order-date), Product (p-id, name, price, description), Product Category (c-id, name). User Address contains a-id, city, state, and country. User Payment (p-id, method, amount). Tracking Detail (t-id, order-no, status), and Cart (cart-id, u-id). Draw the ER diagram for this scenario, along with cardinality and participation aspects.

ER diagram of Simple Online Shopping System



Also, refer two lab experiments of ER Modelling.

Relational Model



Terminologies, or Aspects, or Characteristics, or Concepts of the Relation Model

As discussed earlier, a relational database is based on the relational model. This database consists of various components based on the relational model. These include:

- **Relation:** A **two-dimensional table** used to store a collection of data. (Table).
- **Tuple:** Row of the relation.
- **Attribute/Field:** Column of the relation, depicting properties that define the relation.
- **Attribute Domain:** Set of **pre-defined atomic values** that an attribute can take i.e., it describes the legal values that an attribute can take.
- **Degree:** It is the number of columns in the relation.
- **Cardinality:** It is the number of rows in the relation.

- **Relational Schema: It describes the structure of the relation.** It contains the table name, its attribute names, and their types:

TABLE_NAME (ATTRIBUTE_1 TYPE_1, ATTRIBUTE_2 TYPE_2, ...)

For our student relations example, the relational schema will be:

STUDENT(ROLL_NO INTEGER, NAME VARCHAR(20), CITY VARCHAR(30), PHONENO INTEGER, AGE INTEGER)

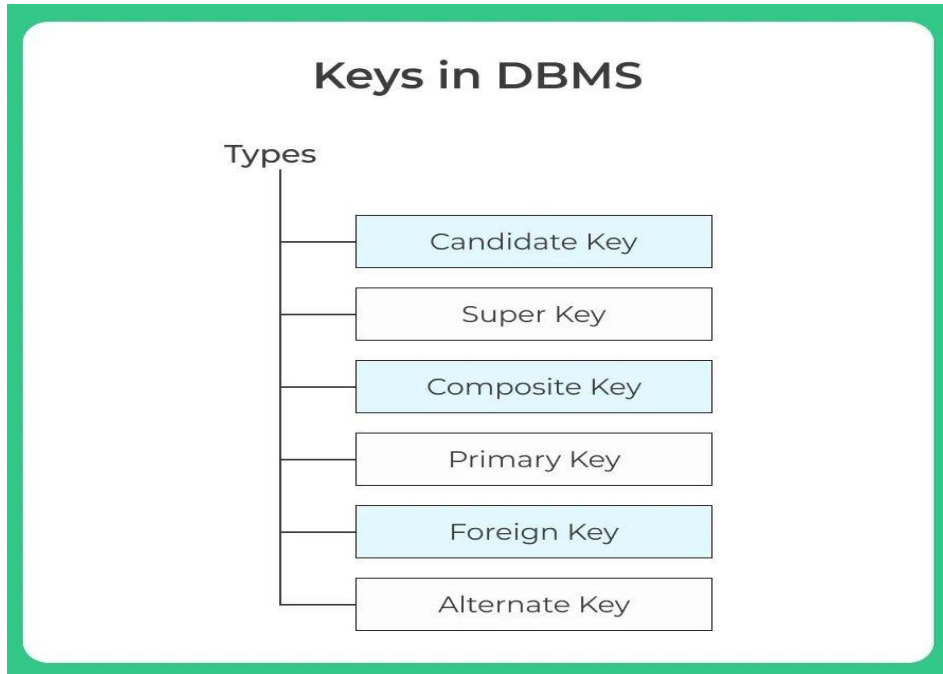
- **Relational Instance:** It is the collection of records, rows, or tuples present in the relation at a given time.

ROLL_NO	NAME	CITY	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI	NULL	18

- **Key:** It is an attribute or a group of attributes that can be used to uniquely identify an entity in a table.

KEYS

A key is **an attribute** or **a set of attributes** that **uniquely identifies a record (or row) within a table**.



STUDENTS

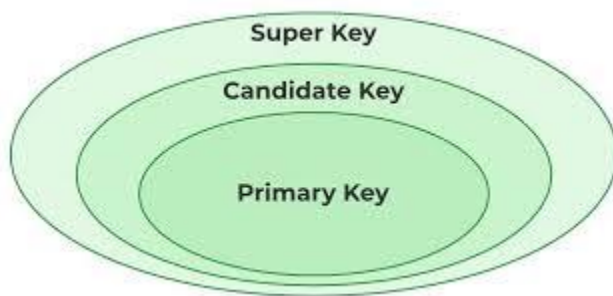
Roll_No	Name	Ph_No	Course	Email	Address
1	Prashant	1234567890	MCA	pra@gmail.com	Mumbai
2	Vipul	0987654321	BCA	vip@gmail.com	Delhi
3	Rohit	5432167890	MCA	Roh@gmail.com	Punjab
4	Prashant	4321567890	B.tech	pr1@gmail.com	Mumbai
5	Rohit	3214567890	MCA	Ro2@gmail.com	Mumbai

©TheStudyGenius.com

- Roll_no, Ph_no, Email are keys. **SINGLE ATTRIBUTE KEYS**.
- Name+Ph_no is also a key. **COMPOSITE KEY**. It is a combination of **MORE THAN ONE ATTRIBUTE** serving as a **KEY**.
- All the above are called **CANDIDATE KEYS**.
- One of the **CANDIDATE KEYS** is taken as the **PRIMARY KEY**. {Roll_no}.

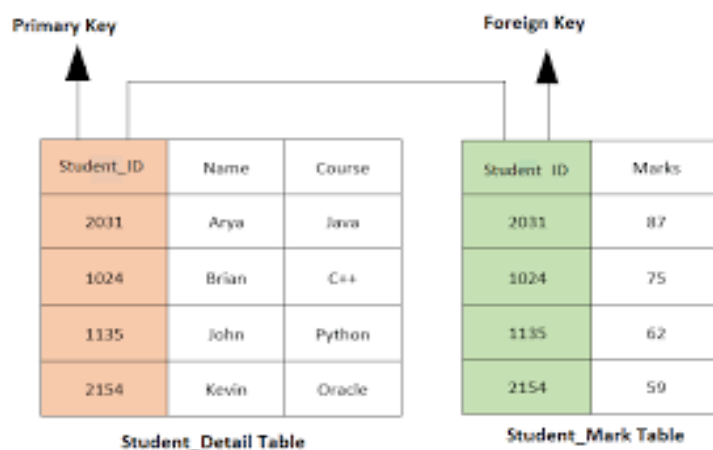
Then all others (Ph_no, Email and Name+Ph_no) become **ALTERNATE KEYS**.

- A super set of a key is called a **SUPER KEY**.
{Roll_no,name}, {Ph_no,Address} and so on are called **SUPER KEY**.

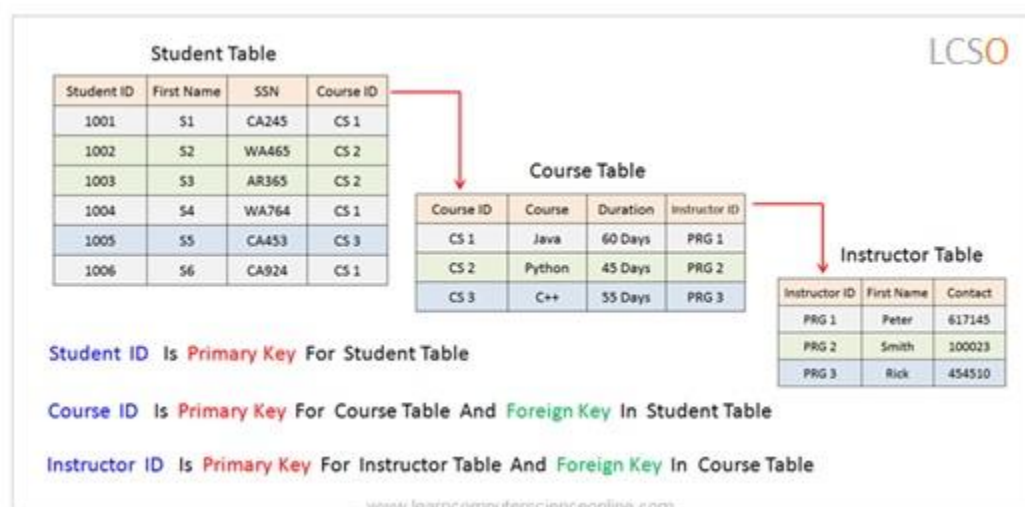


Foreign Key: It is a common column in both tables that joins the records of two tables. The common column should be the primary key in one table and a foreign key in the other table. Foreign key values are to be drawn from the corresponding primary key.

Example 1 on Foreign Key



Example 2 on Foreign Key



Exercise on KEYS

TABLE: Employee

empld	name	address	DOB	job	salaryCode	deptId	manager	schemeld
E101	Keita, J.	1 high street	06/03/76	Clerk	S1	D10	E110	S116
E301	Wang, F.	22 railway road	11/04/80	Sales person	S2	D30	E310	S124
E310	Flavel, K.	14 crescent road	25/11/69	Manager	S5	D30		S121
E501	Payne, J.	7 heap street	09/02/72	Analyst	S5	D50		S121
E102	Patel R.	16 glade close	13/07/74	Clerk	S1	D10	E110	S116
E110	Smith, B.	199 London road	22/05/70	Manager	S5	D10		S121

Exercise 1 on Foreign Key

Student

Roll no.	Student name
1	Amit
2	Priya
3	Vinit
4	Rohan
5	Smita

Performance

Roll no.	Subject code	Marks
1	A	86
1	B	95
1	C	90
2	A	89
2	C	92
3	C	80

Exercise 2 on Foreign Key

Question 11

Write queries (a) to (d) based on the tables EMPLOYEE and DEPARTMENT given below:

teachoo

Table: EMPLOYEE

EMPID	NAME	DOB	DEPTID	DESIG	SALARY
120	Alisha	23-Jan-1978	D001	Manager	75000
123	Nitin	10-Oct-1977	D002	AO	59000
129	Navjot	12-Jul-1971	D003	Supervisor	40000
130	Jimmy	30-Dec-1980	D004	Sales Rep	
131	Faiz	06-Apr-1984	D001	Dep Manager	65000

Table: DEPARTMENT

DEPTID	DEPTNAME	FLOORNO
D001	Personal	4
D002	Admin	10
D003	Production	1
D004	Sales	3